
# Crypto Currency Trading Simulator

Estelle Oliveira
Penda Thiao
Patrick Nsundi Mboli
Julien Rigal

Calcul Haute Performance et Simulation
Master1 / Semestre1

20 Décembre 2024

***Encadrant:***
**Hugo Bollore**: hugo.bollore@uvsq.fr
**Jäsper Salah IBNAMAR** : mohammed-salah.ibnamar@uvsq.fr

**Abstract**

Ce projet aura pour objectif de mettre au point un programme qui nous permettrai de simuler l'évolution d'une (ou plusieurs) crypto-monnaie(s), ainsi que de créer un serveur sur lequel pourrait s'effectuer du trading.

Tout d'abord nous créérons une version simplifiée qui n'implémentera pas les intéractions avec utilisateur, puis la plateforme gagnera en complexité et fonctionnalité. Le projet s'effectuera principalement en langage de programmation C++.

# Contexte et Objectifs

Le projet *Crypto Currency Trading Simulator* a pour but **d'exploiter nos connaissances en programmation et des outils mathémtiques nécessaires** afin de développer un serveur adéquat. Ce projet peut se découper de la sorte :

- Objectif 1 : Créer la crypto-monnaie, et définir son évolution.
- Objectif 2 : Mettre en place des bots aux comportements divers, définir les actions qu'ils pourront réaliser.
- Objectif 3 : Faire un serveur qui contient toutes les données nécessaires, et qui gérera le bon déroulement de ces actions..
- Objectif 4 : Générer des simulations (plus ou moins simplifiées) sur l'évolution d'une crypto-monnaie.

Le SRD-BTC est une crypto-monnaie créée spécifiquement dans le cadre de notre Projet Professionnel Numérique. Elle est la première en date, et c'est pourquoi nous avons commencé avec un design relativement simple. Lorsque la complexité du projet progressera, celle des crypto-monnaies suivra.

Comme son nom l'indique, le SRD-BTC est en partie calquée sur les valeurs du bitcoin sur une période de dix ans (pour la simulation en elle-même, nous stockerons seulement une année de donnée). Cependant, au lieu d'un simple copier-coller, il nous a semblé plus intéressant d'ajouter une variation supplémentaire. Nous avons voulu créer une monnaie qui aurait la forme suivante :

$$\textbf{SRD-BTC} = \textbf{constant} \times \textbf{BTC} + \textbf{random\_number}$$

Pour ne pas arriver sur des valeurs trop chaotiques, nous avons défini ce nombre aléatoire comme étant également fonction de la valeur du bitcoin. Cela nous donne finalement :

$$\textbf{SRD-BTC} = (\textbf{constant} + \textbf{random\_number}) \times \textbf{BTC}$$

À présent, il nous fallait choisir les valeurs que prendraient (ou pourraient prendre) les deux termes dans la parenthèse. Pour cela, nous avons étudié les variations journalières du bitcoin. L'idée était d'avoir une crypto-monnaie qui, grâce à son membre aléatoire, oscillerait autour de la valeur exacte du bitcoin.

Nous voulions une oscillation assez fine, qui pourrait fournir un encadrement suffisamment précis. En la choisissant de cette manière, nous nous projetions dans un futur hypothétique du projet. En effet, en capturant ainsi la valeur d'une monnaie, il devient possible de spéculer plus finement quant aux valeurs futures de cette dernière.

Nous avons donc comparé les variations journalières, et observé que sur dix années la variation maximale avait été de 5.11%. Notre formule devenait donc :

$$\textbf{SRD-BTC} = (0,9489 + \textbf{rand}(0, 2 * 0.0511)) * \textbf{BTC}$$

Ce qui nous donne l'encadrement suivant :

$$0.9489 * \textbf{BTC} <= \textbf{SRD-BTC} <= 1.0511 * \textbf{BTC}$$

## Difficulté : l'implémentation par seconde

Une fois cela fait, nous avions un encadrement journalier de la valeur du SRD-BTC. Cependant, il y avait encore un problème. Notre serveur s'actualisait chaque seconde, et par conséquent, la valeur de notre monnaie également. étant donné que le SRD-BTC était fonction de la valeur du bitcoin, c'était par là qu'il fallait commencer.

Pour éviter d'avoir une simple croissance (ou décroissance) linéaire sur la journée, il fallait trouver un moyen de définir sa valeur seconde par seconde. La tâche devenait alors plus arbitraire. En effet, nous avons pu trouver les données sur la valeur journalière du bitcoin sur dix ans, mais il n'y a pas de tableau qui contiendrait sa valeur à chaque seconde de ces presque 4000 journées.

Nous avons donc fait appel à la fonction random (rd), toujours dans cette optique de ne pas autoriser des variations trop importantes, invraisemblables. À chaque seconde, la valeur est incrémentée de la variation journalière divisée par le nombre de secondes dans une journée, à laquelle on ajoute plus ou moins un nombre aléatoire. Ce nombre aura la valeur absolue suivante :

$$BTC(t+1) = increm + (rd(0, range) + 0.9) * BTC(t)$$

avec

$$range \leq 0.2$$

et

$$\lim_{t \to 0} range = \lim_{t \to 86400} range = 0$$

La variation ne dépassera jamais 10%, et elle convergera vers 0% à mesure que l'on s'approche de la valeur journalière suivante. Sa limite est calculée pour un temps t qui tend vers le nombre de secondes dans une journée, soit 86400.

Une fois l'implémentation par seconde du BTC faite, celle du SRD-BTC suivra simplement la définition de la crypto-monnaie (ajout d'une incertitude de 5.11 pourcents).

Les graphiques suivants illustrent les variations théoriques que nous devrions retrouver dans notre code :

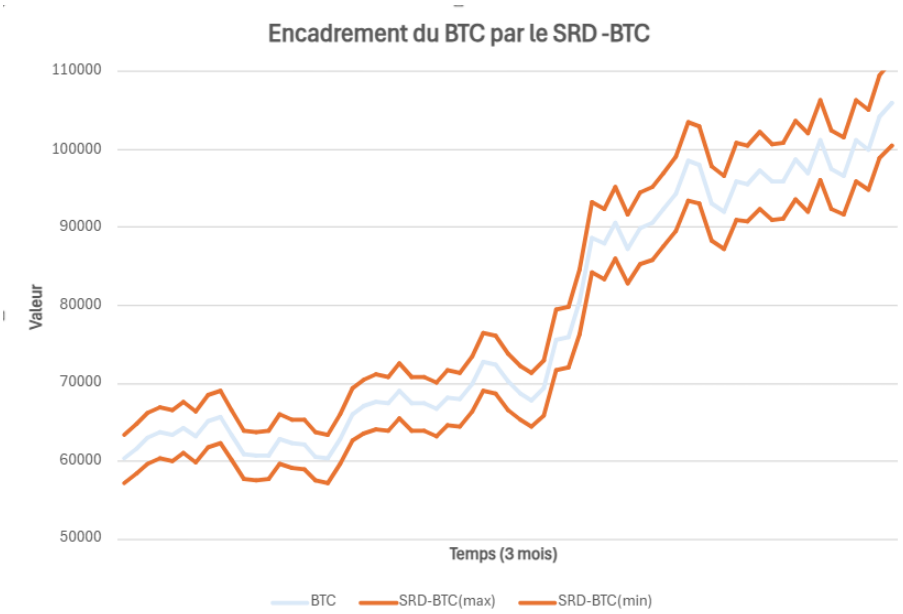


Figure 1: L'encadrement graphique de 5.11% correspondant à l'encadrement numérique.

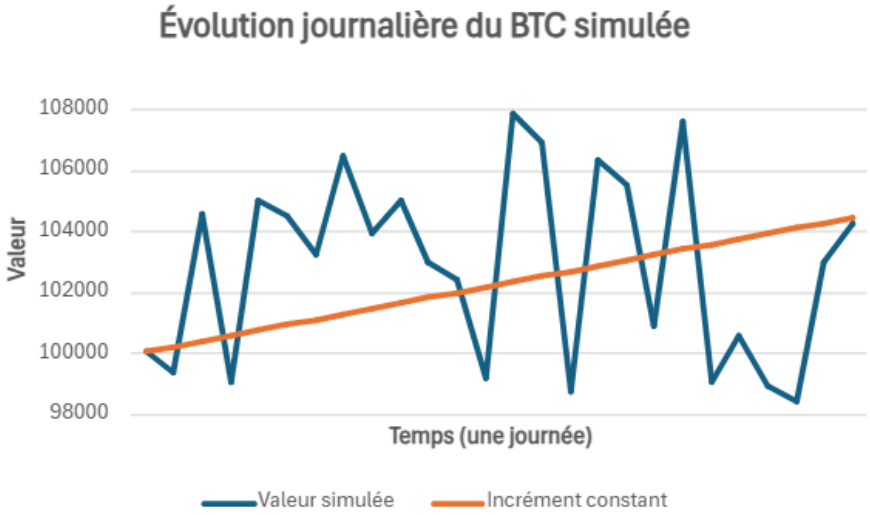


Figure 2: Comparaison entre la courbe ayant été simulée avec une part d'aléatoire, et la courbe que nous aurions obtenu avec un incrément constant.

Une fois la monnaie définie, il nous fallait commencer à réfléchir aux intéraction client/serveur. Dans cette partie, nous allons nous pencher sur la définition des comportement possibles du client. Ensuite nous parlerons des difficultés pour bien implémenter ces comportements dans le cadre du serveur.

Tout d'abord, commençons par préciser que, pour ce premier semestre, le client sera automatisé (comme un bot). Il fera appel à des méthodes de trading déjà implémentées, certaines davantage concentrées sur du court terme, d'autre sur du long terme.

Elles sont pour l'instant au nombre de deux, le "bot_trading" et "bot_investing", et sont pensées de la manière suivante :

### *Bot trading*

Cette méthode est orientée vers de l'achat/revente à plus haute fréquence, et dans de plus amples proportions. Cela se traduira principalement dans l'utilisation des fonctions buycrypto et sellcrypto (dont l'implémentation sera précisée ci-dessous). Le principe est le suivant :

1. Établissement du solde d'origine.
2. Vérification du solde actuel et prise de décision en conséquence:
   - Décision plus risquée si solde actuel $\geq$ solde origin (1).
   - Décision moins risquée sinon (2).
3. Vérification de l'état du marché toutes les cinq minutes.
4. Calcul de l'évolution en comparant les valeurs à $t$ et $t+1$.

5. Si le taux d'évolution dépasse un certain seuil(1.02): `sellcrypto` (à hauteur de 100% du solde pour (1) et 80% pour (2)).
6. Si le taux d'évolution est en deçà d'un certain seuil(0.98): `buycrypto` (à hauteur de 5% du solde pour (1) et 3% pour (2)).

### *Bot investing*

Pour cette méthode, les actions sont similaires, mais effectuées à plus basses fréquences. En effet, elles sont déclanchées par des évènements plus rares (une évolution davantage marquée).

1. Établissement du solde d'origine.
2. Vérification du solde actuel et prise de décision en conséquence:
   - Décision plus risquée si solde actuel $\geq$ solde origin (1).
   - Décision moins risquée sinon (2).
3. Vérification de l'état du marché toutes les cinq minutes.
4. Calcul de l'évolution en comparant les valeurs à $t$ et $t+1$.
5. Si le taux d'évolution dépasse un certain seuil(1.04): `sellcrypto` (à hauteur de 100% du solde pour (1) et 80% pour (2)).
6. Si le taux d'évolution est en deçà d'un certain seuil(0.96): `buycrypto` (à hauteur de 5% du solde pour (1) et 3% pour (2)).

Nous avons pu entrevoir en explicitant ces deux méthodes, qu'il y avait un certain nombre d'actions que le client devait effectuer. Nous allons à présent les lister et les expliquer.

- **getPrice :**

  La fonction **getPrice** permet, au temps t où elle est appelé, d'obtenir l'état du marché. Elle prendra en argument une chaîne de caractères qui devra correspondre à la crypto-monnaie dont on souhaite connaître la valeur.

  Cette fonction a été revue afin de faciliter le trading. En effet, étant donné que le client prend une décision basée sur la comparaison avec la valeur précédente, deux options s'offraient à nous. Il fallait soit créer une fonction **"getpreviousPrice"** qui aurait prit en argument un temps t antérieur et retourné la valeur correspondante, soit tout simplement stocker la valeur retourné à chaque appel de **getPrice** (pour la réutiliser lors d'un appel ultérieur). Nous avons opté pour la deuxième option.

- **getBalance :**

  La fonction **getBalance** est un appel simple qui retourne la valeur du porte-monnaie numérique. Il est possible de choisir sous quelle forme on désire récupérer son solde : la quantité de crypto-monnaie ou la conversion en USD. Cette fonction est utilisée dans l'étape 2 ci-dessus.

  **buyCrypto :**

  Comme son nom l'indique, **buyCrypto** va permettre l'achat d'une crypto-monnaie. La fonction procédera en deux étapes, tout d'abord elle va récupérer la valeur de la crypto-monnaie entrée en argument. Puis elle fera une conversion de la quantité

d'argent investie en une certaine quantité de crypto-monnaie (les quantités seront liées à la valeur retournée par **getPrice**).

**sellCrypto :**

La fonction est presque identique à **buyCrypto** dans sa structure. Simplement la conversion se fait dans l'autre sens (une quantité de crypto-monnaie choisie par le client convertie en USD). La première étape utilisant **getPrice** est inchangée.

*Note : les fonctions buyCrypto et sellCrypto permettront l'actualisation du porte-monnaie du client qui réalise la requête. Le client reçoit ensuite la réponse du serveur si sa requête a abouti, ou bien un message d'erreur signalant le problème.*

Comme dit précédemment, l'implémentation de ces fonctions s'est faite de concert avec les requêtes formulées par le client. Ces requêtes sont envoyées au serveur, gérées par la suite dans une boucle (elle-même implémentée dans la boucle principale du serveur) qui concerne la connexion client.

On pourra dans la suite du projet dédier un thread à la boucle de requête pour permettre au serveur de retourner à l'état d'écoute des connexions clients tout en gérant les requêtes des autres clients, il faudra donc dans l'idée dédier un thread de requête par client connecté.

Ce qui nous amène à la création du serveur et de son architecture.

Maintenant que les actions principales sont crées, nous avons le squelette de notre plate-forme d'échange. Vient maintenant la partie la plus hardue. En effet, dès lors qu'une décision concernant le stockage où le flux de données était prise, il fallait absolument conserver une cohérence forte au risque de devoir transformer une grande partie du code.

Il y a par exemple eu un problème de ce type quand nous avons réalisé qu'il était bien mieux de stocker le porte-monnaie du client à l'intérieur du serveur. S'en est suivi un changement dans toutes les fonctions qui devaient l'utiliser.

Voilà pourquoi il était important de bien conceptualiser notre système. Nous avons donc mis au point un schéma qui représenterait au mieux les flux de données ainsi que les interdépendances :

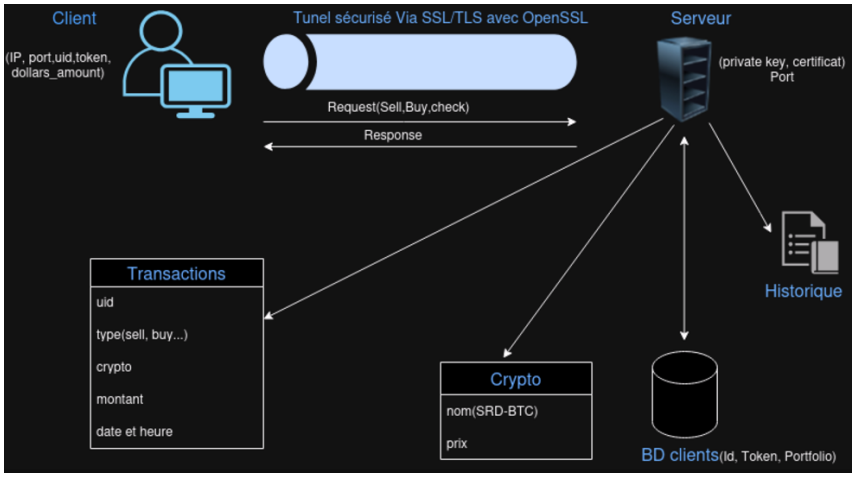


Figure 3: Schéma récapitulatif du fonctionnement du serveur.

Sur la partie haute du schéma, nous pouvons voir ce dont nous avons parlé dans la deuxième partie. Mais une fois que les requêtes sont faites, il faut que le serveur soit en mesure d'y répondre.

### *Un canal sécurisé*

La base de tout, est d'établir une communication sécurisée entre le client et le serveur. Pour cela nous utiliserons SSL/TLS avec OpenSSL.

Le fonctionnement est le suivant : le serveur a une clef privé et un certificat (générés avec OpenSSL donc) qui servent à établir un canal sécurisé entre lui et le client.

### Processus de connexion :

Il y a deux cas de figure lors d'une tentative de connexion. Il peut s'agir de la première connexion, ou bien le client possède déjà un moyen d'identification. **Mais que sont-ils ?**

→ **Couple ID/Token**
L'id est un nombre de 4 chiffres génénéé aléatoire en utilisant la fonction random(). Le token est génénéé avec HMAC, pour ce se faire une chaine aléatoire est générée avec la fonction RAND_bytes d'Openssl, et cette dernière est hachée avec HMAC afin de générer un token unique pour le client.

(pour le futur, à la place de générer une chaine aléatoire, on peut demander à l'utilisateur de fournir un mot de passe, voire même procéder avec clef publique/clé privée et générer un token avec ces informations toujours avec HMAC)

Reprenons donc à l'étape de la connexion. Si le client se connecte sans renseigner de ID/Token (premier cas de figure), il y aura génération d'un ID/Token via la méthode explicité ci-dessus. Tout cela sera stocké dans le fichier "configFile.csv", et ces informations sont évidemment envoyées au client afin qu'il puisse s'identifier la fois suivante.

Pour le second cas de figure, le client doit renseigner l'ID et le token, le serveur récupère ces deux informations et vérifie au niveau du fichier "configFile.csv" que ces informations y soient stockées.

Si oui → la communication est établie.

Si non → *"Authentification failed".*

Une fois le tunnel sécurisé, nous pouvons nous concentrer sur la gestion des requêtes par le serveur.

## Les données :

Tout d'abord, c'est dans le serveur que seront stockées toutes les données utiles. Elles sont de nature diverses.

- Valeur de la monnaie Pour l'instant, nous travaillons avec une crypto-monnaie *fictive*, et donc ses valeurs au cours du temps sont connues à l'avance, stockées dans un fichier "SRD-BTC.csv". Comme vu dans la partie une, ce fichier contiendra les valeurs à chaque seconde sur une période d'un an. À chaque actualisation du serveur, il ira donc chercher la valeur suivante, ce qui mettra également à jour le porte-monnaie du client.

  **Note** : Dans le futur, nous pourrions calquer et suivre l'état du marché réel.

- Informations liées au client Il faut aussi que le serveur conserve toutes les informations importantes liées au client. Les identifiants, les informations contenues dans son porte-feuille, l'historique de ses transactions.

## La gestion des requêtes :

La boucle de requête se situe au niveau du serveur et repose sur la reconnaissance de mot clef. Dans un premier temps, le client va formuler la requête à envoyer au serveur. Le

serveur agit ensuite en fonction, et redirigera vers la méthode concernée lorsqu'il trouvera le mot clef correspondant, menant donc aux fonctions buyCrypto et sellCrypto.

Nous avons réalisé une première approche d'affichage des données des cryptomonnaies. En utilisant gnuplot, nous réalisons une boucle infinie pour obtenir chaque seconde les prix des cryptomonnaie. Le programme écrit ensuite dans un fichier de données, le temps et les prix des cryptomonnaies ; un fichier gnuplot nous permet de reformater le temps en format heures:minutes:secondes, et de retracer le graphique toutes les secondes, à l'aide d'une boucle infinie et d'une pause. On obtient alors un graphique des prix des cryptomonnaies qui s'actualise toute les secondes automatiquement.

Nous avons implémenté cette fonctionnalité dans un thread, ce qui permet de laisser continuellement l'actualisation des données du SRD-BTC se faire tant que le serveur est allumé. L'affichage peut également se faire dans un terminal adjacent.

# Implémentations possibles pour le futur

Pour le second semestre, la complexification du projet pourra s'organiser de la façon suivante :

- Multiplication des clients, des monnaies disponibles. Pour cela, il nous faudra commencer à implémenter du parallélisme, gérer le souci des requètes simultanées, envisager potentiellement l'ajout d'autres "sous-serveurs" qui pourraient traiter certaines demandes de leur côté.
- Affiner le modèle des crypto-monnaies, renforcer la cryptographie. Réfléchir à l'utilisation d'outils mathématiques plus sophistiqués(stochastique), et réfléchir à une meilleure façon de simuler l'évolution d'une crypto-monnaie.

  Pour la partie cryptographie, nous pourrions nous pencher sur la notion de blockchain. Réfléchir à complexifier le chiffrement/cryptage.

# Références

Les sources utilisées pour ce projet sont les suivantes :


- Andreas M. Antonopoulos and David A. Harding, *Mastering Bitcoin: Programming the Open Blockchain (3rd Edition)*, 2023.
- Developer Guides — Bitcoin
- Transport Layer Security sur Wikipedia